IT'S A SMALL WORLD AFTER ALL

OR IS IT? RANDOM GRAPHS, CLUSTERING, AND SMALL-WORLD NETWORKS

AUTHORS: TRAVIS MALLETT
& ERIC FOSSUM

ABSTRACT: It often happens that we heard the phrase, "It's a small world!" after some coincidence meeting. Although mostly thought of in the context of people being connected, the small-world concept has far-reaching implications that seem to be an underlying part of nature itself. Small-world networks are all around us. Not only do we observe natural occurrences of small-worlds, but we are in fact, part of one! Throughout history, it has often been noted how small the world is and, indeed, today, the topic is a popular reminder of how connected we are, especially in this digital-age. This paper presents the history and background of small-world networks and uses computer simulations to gain a fuller understanding about the properties of small-world networks.

TABLE OF CONTENTS

SEC. 1 – INTRODUCTION AND BACKGROUND 2

HISTORY 2
APPLICATIONS 2
MATHEMATICAL SIGNIFICANCE AND BACKGROUND 3

SEC. 1.1 – PREVIOUS SMALL-WORLD STUDIES AND RESULTS 3

MILGRAM'S EXPERIMENT 3
WATTS AND STROGATZ 4

SEC. 2 - COMPUTER SIMULATIONS OF "SMALL-WORLD" NETWORKS 4

ALGORITHM TIME-COMPLEXITY AND SPACE CONSTRAINTS IN A FULL-SCALE MODEL 5
OUR SOLUTION 5
PROGRAM DESIGN 5
C++ Computer Code 6
mknode Function 6
Connlist Function 7
ConnNode Function 7
Find Node Function 9
dellist Function 10
Dijkstra's Algorithm Function 11
Set Friends Function 11
Get Smallest Node Function 12
Main Function 13
RESULTS OF COMPUTER SIMULATION 14
Analysis & Speculations 15

BIBLIOGRAPHY & REFERENCES 17

TABLE OF FIGURES

FIGURE 2: SMALL-WORLD NETWORK & CLUSTERS 3
FIGURE 1: MILGRAM'S EXPERIMENT (LETTER CHAIN) 3
FIGURE 3: CONNECTED CAVEMAN CONSTRUCTION 4
FIGURE 4: DIAMETER VS. CONNECTIONS 14
FIGURE 5: DIAMETER VS. POPULATION 15
FIGURE 6: 3D GRAPH OF DIAMETER VS. CONNECTIONS & POPULATION 15
FIGURE 7: AVERAGE PATH LENGTH (WATTS AND STROGATZ EQUATION) 16

IT'S A SMALL WORLD AFTER ALL

OR IS IT? RANDOM GRAPHS, CLUSTERING, AND SMALL-WORLD NETWORKS

SEC. 1 – INTRODUCTION AND BACKGROUND

HISTORY

We have all had those seemingly impossible, coincidental encounters with people whom we know or are acquaintances of someone we know. These encounters often result in the comment, "Wow! It's a small world!" The concept of a small-world has long been the object of popular fascination and scientific inquiry. In his volume of short stories titled *Everything is Different*, the Hungarian author Frigyes Karinthy speculated that everyone in the world could be connected to anyone else in a chain of no more than five intermediaries (Karinthy 1929). It was not until 1967 when Stanley Milgram conducted his famous small-world experiment. This experiment consisted of selecting arbitrary individuals in the US and giving them a letter with instructions to deliver it to a single target person by mailing the letter to an acquaintance whom the subject deemed closer to the target. Of the letters that reached the target person, the average number of "connections" was 5.2 (Milgram, Travers 1969). Strikingly close to the number predicted by Karinthy, the result continues to be an object of fascination and amusement today. The concept was popularized in the 1990s by John Guare's successful play *Six Degrees of Separation*.

Not only applying to global social studies, the small-world concept is dramatized in other, more popular venues. It is said that Kevin Bacon is the center of the film universe and that all movie actors and actresses (dead or alive!) can be linked to Bacon within 4 other actors or actresses. In 1994, three Albright College students created a game called "The Six Degrees of Kevin Bacon." This trivia game is based on the small-world phenomenon and requires that players try to connect any actor to Kevin Bacon as quickly as possible (Watts 2004).

Every actor is assigned a "Bacon Number." Bacon himself has a Bacon Number of 0. Actors who have appeared in the same film as Bacon are assigned Bacon Numbers of 1. Other actors who have been in the same movies as those with Bacon Number 1 are assigned Bacon Number 2. It was hypothesized that no actor or actress in *any* movie ever has a Bacon Number greater than 4. This hypothesis has become quite a sensation in popular culture as people try to find the shortest paths from any celebrity to Kevin Bacon.

APPLICATIONS

But the study of the small-world concept and the "Six Degrees of Separation" has much deeper implications than simply the sociological curiosity about people and acquaintances. Small-world networks are of much interest in telecommunications, the study of spreading diseases, neural networks, and even geophysics (Watts 1999). Real-world phenomena such as the global economy, food webs, ecosystems, road maps, and the Internet can all be represented as networks which exhibit small-world characteristics or behavior (Watts 2004).

A more profound, underlying structure is implied in the study of small-world networks. Since many natural occurrences are in essence, small-world networks (or can be modeled as such), the study and further exploration of the concepts relating to small-world networks have significance beyond immediate, specific applications.

MATHEMATICAL SIGNIFICANCE AND BACKGROUND

In mathematics, the study of small-world networks falls under the realm of graph theory. In the classic example of modeling networks of people, each person is represented as a vertex while two people knowing each other is represented as an edge between the two respective vertices. By adding vertices and edges appropriately, many problems and real-life situations can be modeled using graphs. But what makes a network or graph that exhibits a sort of "small-worldness" unique from other kinds of graphs such as complete, k-partite, or even completely random graphs?

There seem to be two predominant characteristics which define a "small-world" network—average path length / diameter and clustering coefficient. The most important property of a small-world network is that the distance between any two nodes is, on average, very small (this is in fact why they are called "small world networks!"). In graph theory, the maximum distance between any two vertices (out of all vertices) is referred to as the diameter $D(G)$ of the graph. The hypothesis of the "Six Degrees of Separation" concept is that for a graph G representing the network of all people on Earth and their connections, $D(G) \leq 6$. Secondly, small-world networks (found in real life) tend to have many smaller of tightly-woven subgraphs called clusters which are connected to each other (See figure).

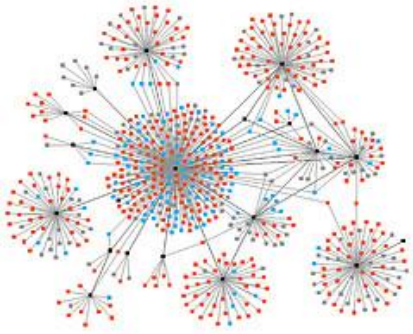

Figure 2: Small-world Network & Clusters

In other words, the clustering concept refers to the fact that we expect many of our friends are also friends with each other. The *clustering coefficient,* $C(G)$ is a measure of the underlying graph structure. An interpretation of the clustering coefficient is the probability that two vertices (u, v) will be connected, given that each is also connected to a "mutual friend" (w) (Watts 1999). The study of small-world networks in an abstract sense is an important part of graph theory today. These concepts of diameter and clustering are part of many "every-day" graph theory problems.

SEC. 1.1 – PREVIOUS SMALL-WORLD STUDIES AND RESULTS

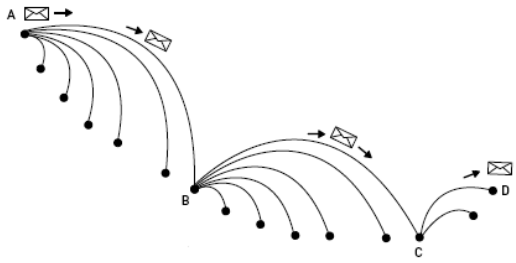

Figure 1: Milgram's Experiment (letter chain)

MILGRAM'S EXPERIMENT

When Milgram carried out his experiment his results were fascinating and important because they represented empirical evidence rather than abstract constructions. But Milgram's presents a more clouded picture after further investigation. Milgram arbitrarily selected a set of 296 people in Nebraska and Boston that are asked to start the chain letters to be sent to the target person Boston. However, only 64 of the letters reached the target. Milgram's theory and experiment has undergone harsh criticism, especially since Milgram never followed up on the lost letters. Additionally, since all the letters were to be sent within the US, there is not much that can be said about a "global network." Although Milgram took a first step in the important study of the small-world phenomenon, he wasn't able to provide any satisfactory results. Rather, his results were somewhat dramatized due to the fact that they were indeed so close to the prediction almost 50 years earlier by Karinthy.

WATTS AND STROGATZ

Analyzing small-world network graphs is often done in two ways, viz., a completely random graph is used as a basis for a structured, small-world, but slightly random, graph is used. The advantages of the first are simple; random graphs are easily obtained and one characteristic of random graphs is that they usually have very low diameters, meaning that any two nodes in the graph are closely-connected, like in a small-world network. But random graphs have a much lower clustering than real-life small-world phenomena (Franceschetti 2006). A refinement on modeling small-world networks with random graphs has been proposed by Watts and Strogatz. They created a model that is the superposition of a structured subgraph of "local contacts" and a random subgraph of "long-range contacts" (Watts and Strogatz 1998). It was noted that by adding a small number of edges uniformly at random to a structured subgraph like a ring or a mesh, it is possible to drastically reduce its diameter (Franceschetti 2006). For an example of a clustered, sparse graph, refer to the figure. This is what is commonly referred to as the *caveman graph* which consists of $\frac{n}{k+1}$ isolated cliques or "caves" where $(n = order, k = avg(deg(V(G)))$. These cliques are all then connected to each other.

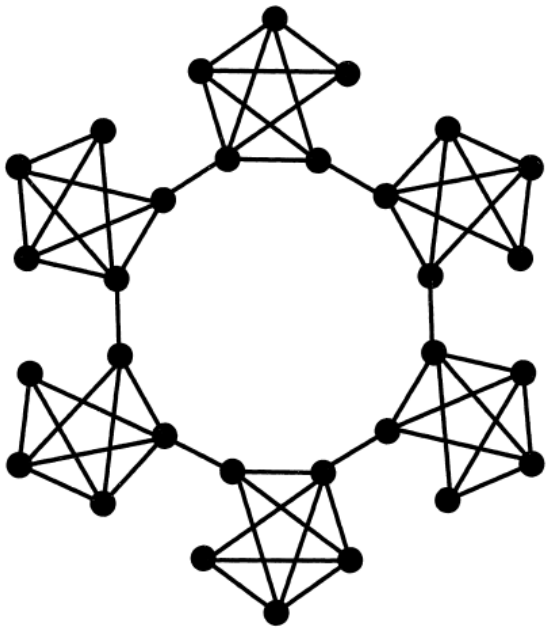

Figure 3: Connected Caveman Construction

Although they studied more structured graphs such as the Caveman graph, Watts and Strogatz were able to derive some important information about random graphs in general. They calculated that the average length from one node to another in a random graph of order *n* and average vertex degree is:

$$L_{random} \sim \frac{\ln(n)}{\ln(k)}$$

And the clustering coefficient of a random graph is:

$$C_{random} \sim \frac{k}{n}.$$

Using Watts' and Strogatz's results, it is simple to calculate an estimate for the average number of connections required to meet the Six Degrees of Separation hypothesis.

$$6 = \frac{\ln(6.8 * 10^9)}{\ln(k)} \xrightarrow{yields} k = 44$$

Clearly an average of 44 connections per person on Earth is a reasonable number and perhaps, much on the conservative side. Obviously since this is the *average* number of connections and we used 6 as the *average* length, obtaining information about the diameter (maximum length) of this graph would require a probability distribution for number of connections that would be pure speculation. However, these results from Watts and Strogatz provide a useful starting place as parameters in a computer simulation.

SEC. 2 - COMPUTER SIMULATIONS OF "SMALL-WORLD" NETWORKS

The computer simulation relevant to this paper deals with the specific small-world network of the people-connections on earth. The current population on Earth is reported to be 6.8 billion people. Thus, there are some inherent difficulties with a "brute force" method to build a graph model of all 6.8 billion people and calculate the diameter of such.

ALGORITHM TIME-COMPLEXITY AND SPACE CONSTRAINTS IN A FULL-SCALE MODEL

Unfortunately, there is a minimum information bound that restricts our ability to handle a full-scale model. For example, if we were to *just* create an adjacency matrix for the entire graph, we would have to create a $(6.8 * 10^9)$ x $(6.8 * 10^9)$ matrix. This yields $4.6 * 10^{19}$ bits of information (at least) that would be required. This means that we would require 5.75 Exabytes (billions of gigabytes) to store just the adjacency matrix! According to wolframalpha.com, this represents almost half the estimated information content of *all* human knowledge! In addition to storage constraints dealing with a full-scale model, there are time constraints as well. Dijkstra's Algorithm alone represents over an order n^2 complexity to calculate the eccentricity of a single node. Since there are *n* nodes in the small-world, this brings the total complexity to order n^3. Addition steps in the process eventually lead to a total of order n^5 complexity (see code in following section). With $n = 6.8\ billion$, the number of cycles required to compute the diameter is $1.4\text{x}10^{49}$. Using the world's fastest supercomputer to date[1], this would take about 4.439 septillion (10^{24}) years to compute.

OUR SOLUTION

Rather, our solution is to simply create a random graph (which will provide the desired low diameter like a real small-world network) and calculate how the diameter changes based on a) the order of the graph, b) the average number of connections per vertex, and c) the clustering coefficient. By using smaller population sizes, we hope to get several data-points that can relate how the *n, k, and C* affect the diameter of a graph. It is hoped that sufficient information might be obtained to extrapolate the results to approximate a figure for a graph of order 6.8 billion.

[1] Tianhe-I China Super Computer operates at 2.057 Petaflops.

Our program will essentially run an arbitrary number of "tests" by creating random graphs of various orders and average connections. The final results are printed to the screen for each graph. Given enough data, we will essentially obtain points for a 2-variable function $D(n, k)$. Plotting this function and extrapolating it will hopefully reveal insights about how our small-world network on Earth works.

PROGRAM DESIGN

The program is built in several smaller chunks that are described in the coding section to follow. The order of flow is:

1. Initialize *n* nodes
2. Randomly connect each node with *k* connections.
3. Calculate the eccentricity for each node using a variant of Dijkstra's Algorithm.
4. Record the maximum eccentricity in the graph (which is the diameter).
5. Return results.
6. Repeat process for next inputs of *n* and *k*.

The process for randomly connecting the nodes is as follows. Given a 1-regular "ring" graph of order *n* (this is to ensure that the graph is connected), follow these steps:

1. Choose a starting node
2. Follow the pointer from that node (recall the ring) around the graph a random number of nodes. The program will now be pointing at node V_{rand}.
3. Create a connection from the starting node to each in $[V_{rand}, V_{rand+1} + \cdots + V_{rand+k}]$.
4. Increment starting node to the next node.
5. Repeat process until all nodes have been process. Ignore duplicate node connections (retry with a different random number).

C++ COMPUTER CODE

The following section contains the complete C++ code that implements the design previously described. The program was written in many small sections called functions that each perform specific tasks. The "main" function at the end shows the overall flow of the program.

mknode Function

The first and most essential function initializes a number (given by the variable "size") of vertices in memory.

```
Node* mkNode(Node **head, long name)
{
   Node *temp;
   int i;

   temp = new Node;
   if( temp == NULL )
   {
      // Memory Error
      cout << "MEMORY ERROR!!!" << endl;
      return NULL;
   }
   int size = sizeof(temp->friends) / sizeof(long);
   for ( i=0; i<size; i++ )
      temp->friends[i] = NULL;

   if ( *head == NULL )
   {
      temp->name = name;
      temp->prev = temp;
      temp->next = temp;
   }
   else
   {
      // Construct temp
      temp->name = name;
      temp->next = *head;
      temp->prev = (*head)->prev;

      // replace head node
      Node *prevTail = (*head)->prev;
      (*head)->prev = temp;
      prevTail->next = temp;
   }
   *head = temp;
   headNode = temp;
   return(*head);
}
```

Connlist Function

The connList function has the end effect of assigning *connections* number of edges, randomly between the nodes. This function loops over every node in the network and chooses a random set of edges to connect.

```
void connList(Node **node)
{
   //threader(node);
   long counter = (*node)->name;
   do
   {
      connNode(node);
      *node = (*node)->next;
   } while( ((*node)->name) != counter );
   return;
}
```

ConnNode Function

The connNode function is called by connList and contains the code for actually assigning the edges randomly to a node. These connections

```
void connNode(Node **node)
{
   int i, j, newGuysName, headNull = 0;

   // Init.
   globalNode = *node;
   for ( i=headNull; i<connections; i++)
   {
      currentFriends[i] = 0;
      array[i] = 0;
   }

//    cout << (*node)->name << " -> ";
   while ( (headNull<connections) && ((*node)->friends[headNull]!=NULL) )
   {
      newGuysName = (*node)->name - (*node)->friends[headNull]->name;
      if ( newGuysName < 0 )
         newGuysName = (headNode->name + 1) - (-1 * newGuysName);
      // insert sort
      long temp;
      j = 0;
      while ( (newGuysName > array[j]) && (array[j] != 0) )
         j++;
      do
      {
         temp = array[j];
         array[j] = newGuysName;
         newGuysName = temp;
         j++;
      } while ( newGuysName != 0 );
                                        ……….
```

```
                                        ……….
//      cout << (*node)->friends[headNull]->name << " : ";
      headNull++;
   }
   for ( i=0; i<headNull; i++)
   {
      currentFriends[i] = array[i];
   }

   // Insert sort
   j = headNull;
   i = headNull;
   while ( array[connections-1] == 0 )
   {
      long newGuy;
      Node* freeNode = NULL;
      freeNode = findNode();

      if( freeNode==NULL )
         break;
      newGuy = (*node)->name - freeNode->name;
      if ( newGuy < 0 )
         newGuy = headNode->name-((-1*newGuy)-1);

      j = 0;
      while ( (newGuy > array[j]) && (array[j] != 0) )
         j++;

      if ( (array[j] != newGuy) && (newGuy != 0) )
      {
         // Bubble and increment
         long temp;
         do
         {
            temp = array[j];
            array[j] = newGuy;
            newGuy = temp;
            j++;
         } while ( newGuy != 0 );
      }
   }

   // connect
   long itr = 0;
   Node *tmpNode = *node;
   for ( i = 0; i != connections; i++ )
   {
      // already connected?
      bool alreadyFriend = false;
      if (array[i] == 0)
         break;
      for(j = 0; !alreadyFriend && (currentFriends[j]!=0) && (j!=connections); j++)
         if(array[i] == currentFriends[j])
            alreadyFriend = true;
                                        ……….
```

```
                                        ……….
  if (!alreadyFriend)
      {
         itr = array[i] - itr;
         int counter;
         // find friend
         for (counter = itr; counter > 0; counter--)
            tmpNode = tmpNode->next;
         // add friend
         (*node)->friends[headNull] = tmpNode;
         headNull++;
//          cout << tmpNode->name << "* : ";
         // double link
         counter = 0;
         while( (tmpNode->friends[counter] != NULL) && (counter != connections) )
            counter++;
         tmpNode->friends[counter] = *node;
      }
   }
//   cout << endl;

   return;
}
```

Find Node Function

The findNode function searches for a node in the population that the current node can be connected to that isn't already "full." If no such node exists, then the graph is completely connected.

```
Node* findNode()
{

   long counter = globalNode->name;
   long name = rand() % population;
   Node *freeNode = NULL;
   Node *tempNode = globalNode;

   do {
      if( (tempNode->friends[connections-1] == NULL) && (tempNode != globalNode) )
      {
         if( name < 0 )
         {
            long guysName = globalNode->name - tempNode->name;
            if ( guysName < 0 )
               guysName = (headNode->name + 1) - (-1 * guysName);
            int itr = 0;
            while ( (itr<connections) && (array[itr]!=guysName) )
               itr++;
            if ( array[itr] != guysName)
               freeNode = tempNode;
         }
         else if ( name == 0 )
            freeNode = tempNode;
      }
                                        ……….
```

```
                                        ……..
 if( name == -30000 )
        {
           break;
        }
        tempNode = tempNode->next;
        name--;
    } while ( freeNode == NULL );

    return(freeNode);
}
```

dellist Function

After a test has been run, the memory used to store the graph (linked list) must be erased to prevent data corruption in future tests.

```
void delList(Node **node)
{
    Node *temp;
    while ( *node != NULL )
    {
        temp = *node;
        *node = (*node)->next;
        delete temp;
    }
    return;
}
```

Dijkstra's Algorithm Function

This function calculates the eccentricity of a node using a Dijkstra-like algorithm.

```
long dijkstrasNode(Node **node)
{
   Node *temp = *node;
   long diameter = 0;

   long counter = temp->name;
   do {
      temp->distance = population;
      temp->visited = false;
      temp = temp->next;
   } while ( counter != temp->name );

   temp->distance = 0;
   do {
      temp->visited = true;
      long tmp = setFriends(&temp);
      if ( tmp > diameter )
         diameter = tmp;
      temp = getSmallest(node);
   } while (temp != NULL);
   return diameter;
}
```

Set Friends Function

The setFriends function gives a "friend" assignment to a node that has been already "touched" or visited within the Dijkstra's Algorithm.

```
long setFriends(Node **node)
{
   int itr_fri;
   long newDistance = (*node)->distance+1;
   long newDiameter = 0;

   for (itr_fri=0; (itr_fri<connections) && ((*node)->friends[itr_fri] != NULL); itr_fri++)
      if ( (*node)->friends[itr_fri]->distance > newDistance )
      {
         (*node)->friends[itr_fri]->distance = newDistance;
         newDiameter = newDistance;
      }

   return newDiameter;
}
```

Get Smallest Node Function

The getSmallest function returns the node with the least "weight" or distance from the starting node that has been calculated within the Dijkstra's Algorithm.

```
Node* getSmallest(Node **node)
{
   int i;
   Node *temp, *currSmall = NULL;

   // end node
   if ( !((*node)->visited) )
      return (*node);

   // recursive
   // needs to init currSmall
   for ( i=0; i<connections; i++ )
   {
      if ((*node)->friends[i] == NULL)
         break;
      if ( (*node)->friends[i]->distance > (*node)->distance )
      {
         temp = getSmallest(&((*node)->friends[i]));
         if ( currSmall == NULL)
            currSmall = temp;
         else if ( temp != NULL )
            if ( temp->distance < currSmall->distance )
               currSmall = temp;
      }
   }

   // return
   return (currSmall);
}
```

Main Function

The main function will run a number of tests on a set of populations and corresponding numbers of connections. The program will build a random graph starting with population 100 and number of connections per node 5 and calculate the diameter. It will then proceed to calculate the diameter for increasing sets of populations and connections by increasing the population by 200 and connections by 3 each time.

```
int main(int argc, char** argv)
{
   Node *temp, *head = NULL;
   long i;

   int itr;
   int pop;
   long conns;
   for( pop=100; pop<=1000; pop+=200 )
   {
      for( conns=5; conns<=15; conns+=5 )
      {
         // Init...
         setValues(conns, pop);
         for( i=0; i<pop; i++ )
            mkNode(&head, i);

         // connect
         temp = head;
         connList(&temp);

         // find diameter
         temp = head;
         long counter = temp->name;
         int diameter = 0;
         do {
            int value = dijkstrasNode(&temp);
            if ( value > diameter )
               diameter = value;
            temp = temp->next;
         } while ( temp->name != counter );

         cout << "Population: " << pop << " Connections: " << conns << " Diameter: " <<
diameter << endl;

         Node *test;
         head->prev->next = NULL;
         while ( head != NULL )
         {
            test = head;
            head = head->next;
            delete test;
         }
      }
   }
   return 0;
}
```

RESULTS OF COMPUTER SIMULATION

The simulation looped over the following values of order and average connections:

n (Order/Population)100 – 1900 (step by 200s)

k (Avg. Connection Size) 5-23 (step by 3s)

By holding the population size constant, we can graph the diameter versus the number of connections per node for multiple population values:

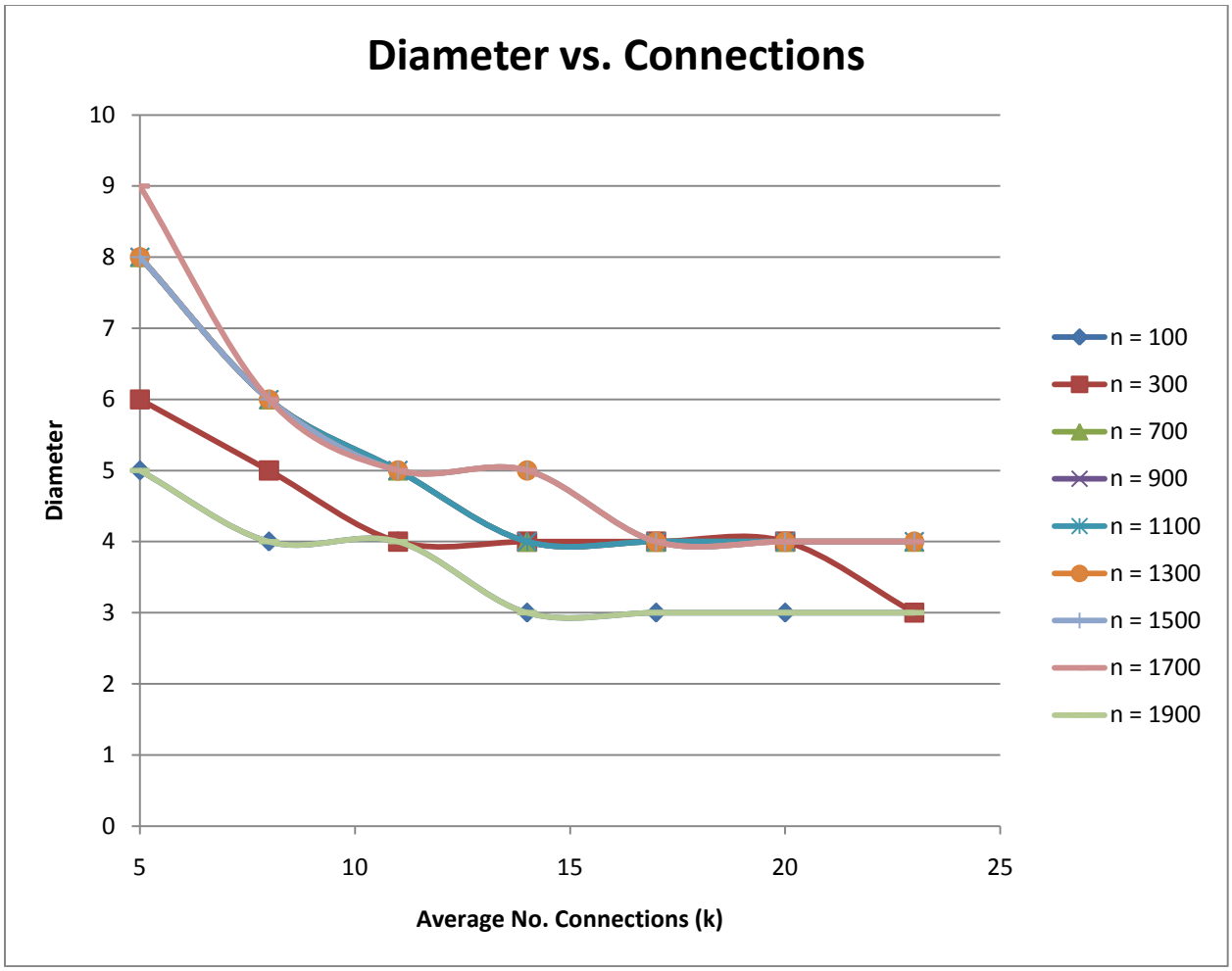

Figure 4: Diameter vs. Connections

Of particular note is the decreasing pattern as the number of connections per node increases. This makes intuitive sense since as the number of connections increases, the longest path between any two nodes should decrease.

Additionally the diameter versus population can be graphed by holding the number of connections per node constant.

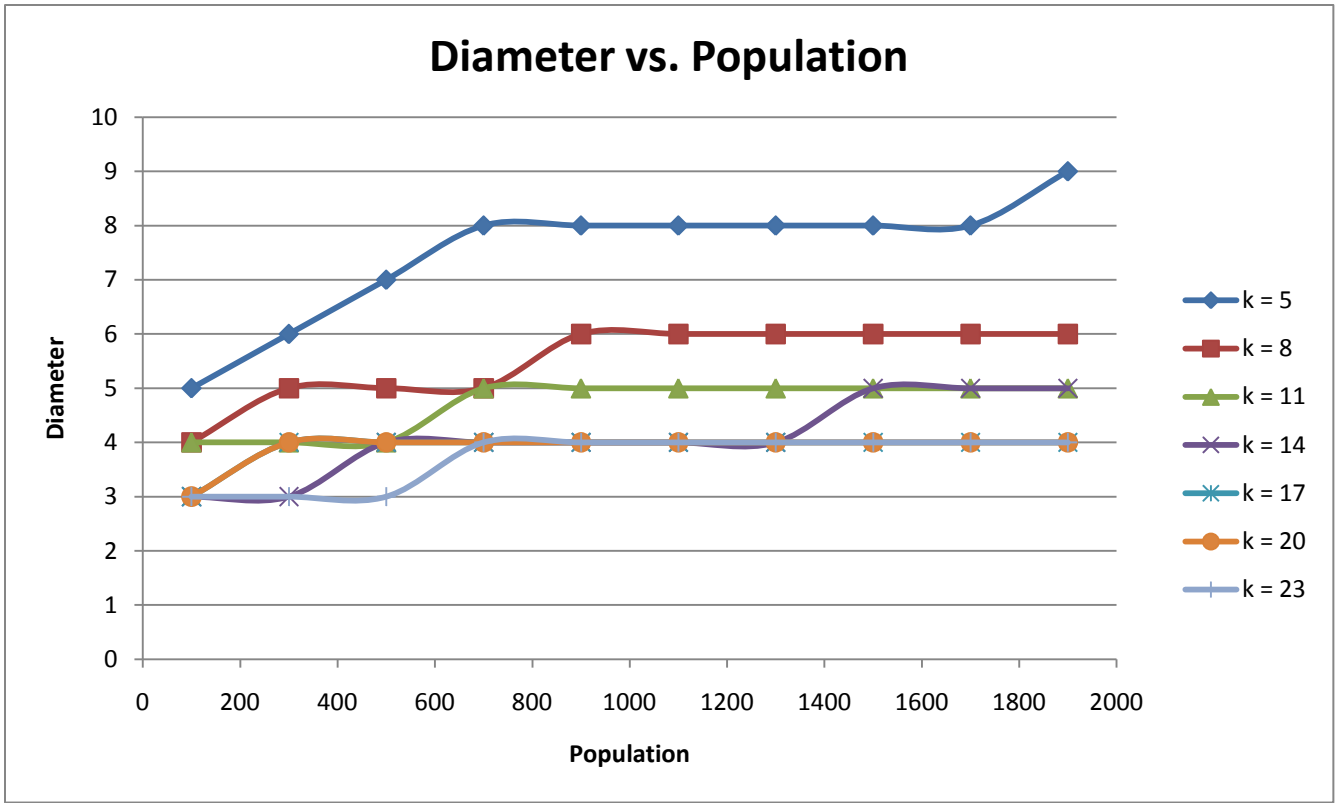

Figure 5: Diameter vs. Population

This graph also makes intuitive sense. Obviously the average (and certainly the maximum) distance from node to node increases as the population increases. However, as the population increases, adding 3 more connections and recalculating the diameter has little or no effect.

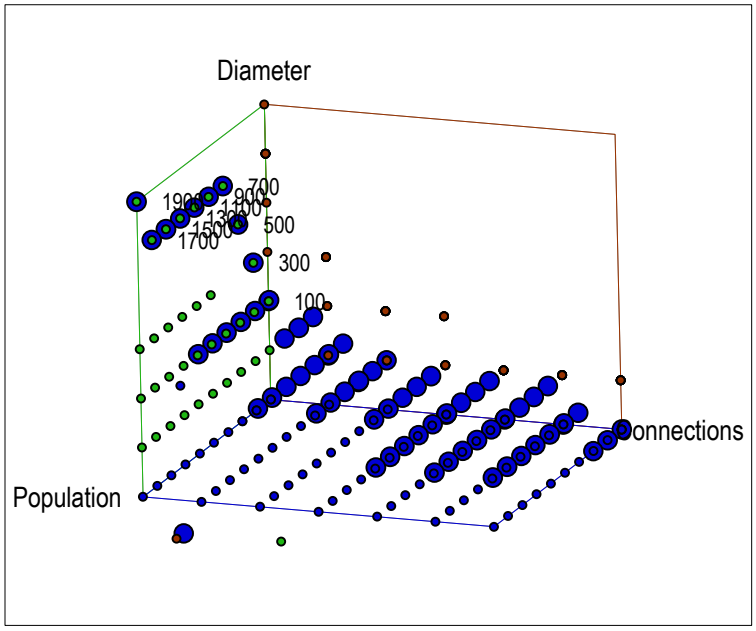

Figure 6: 3D graph of Diameter vs. Connections & Population

Finally, the diameter can be plotted on a 3D graph as a function of both connections and population. The features of both of the previous graphs can be seen in the 3D version.

ANALYSIS & SPECULATIONS

The results of the computer simulation reveal the general properties of the diameter of small-world networks. But without further tests, fails to give enough data to extrapolate to a population of 6.8 billion. Additionally, even if we *could* extrapolate to such a large population value, there still remains the problem of *what* the average number of connections per node should be. If we graph the average path length between two nodes using Watts' and Strogatz's results, we can see that with a population of 6.8 billion, the

average path length approaches 3 degrees of separation for a large number of connections (over 1000). Note that since this is the *average* path length, the diameter will actually be higher than that. It seems likely that given some distribution for connections per person, bounds could be calculated. However, is it likely that the *average* number of connections or friends on earth for *everyone* is less 1000. If this is the case, it also seems likely (without being quantitative here), that the diameter will be greater than 6.

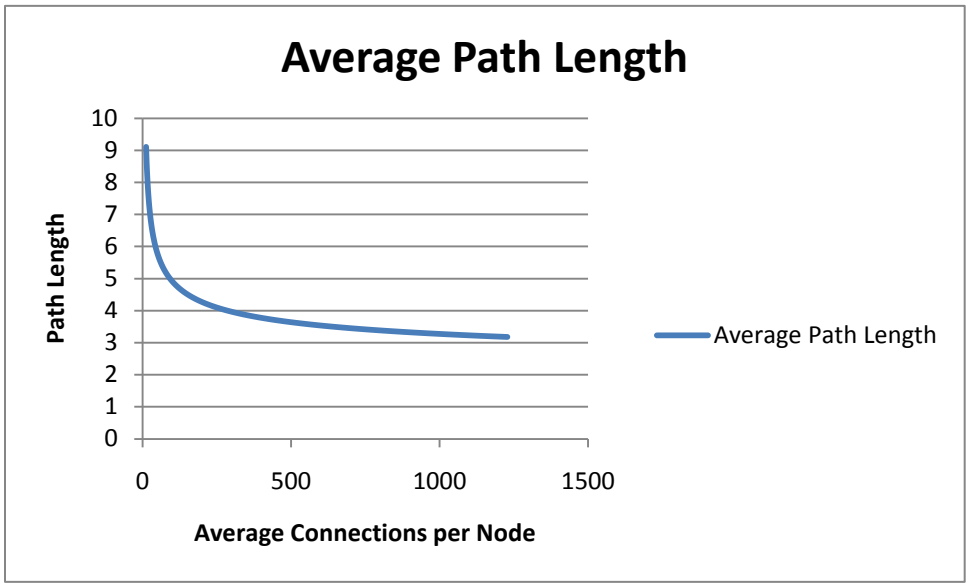

Figure 7: Average Path Length (Watts and Strogatz Equation)

Without further social information regarding the number of connections per person, it seems unlikely that a conclusion about whether the Six Degrees of Separation hypothesis is true. The results of this analysis confirm the nature of diameter in random networks.

Thus the question still remains, is everyone on earth connected within 6 degrees? Or are there two people on earth who's shortest path of connection is much more than that? Without further simulation and resources, further steps in solving this problem (or at least painting a clearer picture) remain in the hands of graph theorists who may yet provide more theoretical results that could lead to a solution.

BIBLIOGRAPHY & REFERENCES

Franceschetti, Massimo, and Ronald Meester. "Short Communications - Navigation in Small-World Networks: a Scale-Free Continuum Model." *Journal of Applied Probability*. 43.4 (2006): 1173. Print.

Guare, John. *Six Degrees of Separation*. New York: Dramatists Play Service, 1992. Print.

Karinthy, F. (1929). *Chains. Everything is Different*. Budapest.

Travers, Jeffrey, and Stanley Milgram. "An Experimental Study of the Small World Problem." *Sociometry*. 32.4 (1969): 425-443. Print.

Watts, Duncan J. "Networks, Dynamics, and the Small-World Phenomenon." *American Journal of Sociology*. 105.2 (1999): 493-527. Print.

Watts, Duncan J. *Six Degrees: The Science of a Connected Age*. New York: W.W. Norton, 2004. Print.

Watts, Duncan J. *Small Worlds: The Dynamics of Networks between Order and Randomness*. Princeton studies in complexity. Princeton, N.J: Princeton University Press, 2004. Print.

Strogatz, SH and Watts, DJ. "Collective Dynamics of 'small-World' Networks." Nature. 393.6684 (1998): 440-2. Print.Bottom of Form